FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Departamento de Engenharia Informática

Trabalho Prático Nº2

The Rise of the Ballz

Diogo Jordão Filipe, José Miguel Silva Gomes, Pedro Tiago dos Santos Marques

FCTUC, Departamento de Engenharia Informática,

Universidade de Coimbra, Portugal

{uc2018288391, uc2018286225, uc2018285632}@student.uc.pt

16, maio 2021

Índice

1 Abstract

No mundo da inteligência artificial, os algoritmos genéticos desempenham um papel importantíssimo no que toca ao desenvolvimento de agentes adaptativos. O estudo que se vai apresentar visa reunir conclusões relativas à evolução e adaptação desses agentes através de redes neuronais e algoritmos genéticos. Para isso, desenvolvemos vários controladores no *Unity* para que o agente *D31* consiga controlar uma bola, defender remates e jogar contra outro agente.

2 Keywords

Agente adaptativo, algoritmo genético, seleção por torneio, pressão seletiva, recombinação, mutação, seleção, parametrização, aptidão.

3 Introdução

3.1 Meta 1

Na primeira meta foram implementadas as funcionalidades básicas do algoritmo genético: recombinação, mutação, seleção, parametrização e aptidão.

Os algoritmos para a mutação gaussiana ("MutateGausssian") e a seleção de torneio ("tournamentSelection") foram implementados tal como foram descritos no enunciado, nos scripts "GeneticIndividual.cs" e "TournamentSelection.cs" respetivamente.

Para a recombinação foi implementada a função "Crossover" no script "GeneticIndividual.cs". Primeiramente gera-se um float random que se for menor que a probabilidade recebida na função, ocorrerá a recombinação dos genótipos, isto é uma forma de tornar este processo aleatório. Para além disso, é gerado um inteiro aleatório com valor máximo igual ao tamanho do genótipo, que irá determinar, caso ocorra a recombinação, até onde no genótipo esta será feita. Para a recombinação em si, apenas é usada uma variável auxiliar para trocar os valores entre os dois indivíduos ao longo dos seus genótipos.

Para a aptidão, foram desenvolvidas duas funções simples para determinar o seu valor, que são usadas em "GetScoreRed" / "GetScoreBlue" no script "D31NeuralControler.cs", e foram executadas para poder analisar a sua capacidade evolutiva, e assim na próxima meta ser possível adaptá-las para resolver os cenários fornecidos. Nas execuções foi usado o cenário "Evolving-ControlTheBallToAdversaryGoal", dado ser o mais simples e geral dos cenários evolutivos, com mutação gaussiana, método de seleção por torneio, e as seguintes configurações:

- 10 segundos de simulação;

- 50 de tamanho da população;
- 100 gerações;
- 5 de tamanho do torneio;
- 15% de probabilidade de mutação;
- 70% de probabilidade de recombinação.

```
1   Function GetScoreRed()
2       fitness ← 0;
3       fitness ← fitness + (1 − average(DistanceToBall));
4       fitness ← fitness + (1 − average(DistanceToAdversaryGoal));
5       fitness ← fitness + (average(AgentSpeed));
6   return fitness;
```

Fig.1 - Função de Fitness inicial para o cálculo da aptidão

A primeira função consiste em usar a média da distância à bola, à baliza adversária e da velocidade do agente. A lógica é bastante simples, queremos que indivíduos que estejam em média mais perto da bola e da baliza adversária (menor distância das mesmas), e que obtenham velocidades superiores, tenham um valor de aptidão mais elevado.

Depois das 100 gerações, podemos usar o ficheiro "best" gerado para observar o comportamento aprendido, neste caso o agente bate apenas uma vez na bola e acelera bastante em direção à baliza adversária. Ao analisar o desfecho pode-se concluir que para conseguir o maior valor de aptidão possível o agente primeiramente aproxima-se da bola, para minimizar a sua distância à mesma, e "chuta-a" para próximo da baliza, seguidamente dirige-se para a baliza adversária para também minimizar essa distância. Ao ter a bola perto da baliza ele consegue estar o mais próximo possível das duas ao mesmo tempo, sempre sem abrandar para maximizar a velocidade. Esta função terá claramente limitações quando se pretender que o agente controle a bola em direção à baliza, pois teria valores mais baixos de aptidão devido à velocidade inferior, e consequentemente, ao demorar mais tempo a chegar à baliza adversária, vai estar em média mais distante da mesma. Porém para um agente que se queira que "remate" a bola (ou seja, apenas um toque), seria bastante útil devido às velocidades atingidas em direção à bola.

```
1   Function GetScoreRed()
2       fitness ← 0;
3       fitness ← fitness + (1 − average(DistanceToBall));
4       fitness ← fitness + (1 − average(DistanceToAdversaryGoal));
5       fitness ← fitness + (average(AgentSpeed));
6       fitness ← fitness * hitTheBall;
7   return fitness;
```

Fig.2 - Segunda função de Fitness desenvolvida

Na segunda função de aptidão desenvolvida apenas se fez uma alteração à anterior, no final de se somar o que já se somava, multiplica-se esse valor pelo número de vezes que o agente tocou na bola. Isto foi uma tentativa de conseguir com que o agente controlasse mais a bola ao invés de a "rematar" e continuar em frente, e embora tenha resultado, ou seja, o agente agora tenta empurrar a bola, este perdeu muita da sua velocidade. Com esta função o agente aproxima-se da bola e começa a empurrá-la muito lentamente, não conseguindo chegar perto da baliza antes de o tempo de simulação acabar, isto deve-se ao facto de ao estar em contacto com a bola, são registados muitos toques (pode registar centenas de toques), logo o agente para maximizar a sua aptidão foca-se apenas em estar em contacto com a bola.

Esta abordagem vai ser claramente útil para cenários em que se queira que o agente controle a bola, porém é preciso arranjar uma solução para o peso enorme que os toques na bola têm no cálculo do valor da aptidão, como por exemplo, limitar o número de toques.

4 Configuração Experimental - Meta 2

Através do uso de redes neuronais, algoritmos genéticos e mecanismos de aptidão denominados de fitness é possível através do papel da experimentação aprender, aperfeiçoar e melhorar a performance do agente D31. Para tal, através do cálculo de um valor de fitness tendo em conta a arquitetura da rede neuronal é concebível obter um bom algoritmo genético.

Após várias experiências e análise dos seus resultados chegámos a um algoritmo de fitness que, fazendo uso dos atributos que compõem a rede neuronal do agente D31, cumprem com o objetivo de marcar mais golos do que o adversário.

Este cálculo da aptidão tem por base o mecanismo de Recompensa/Punição em que caso o agente tenha uma resposta positiva a sua aptidão aumenta e, caso tenha uma resposta negativa a sua aptidão diminui.

4.1 Cena 1: Evolving-ControlTheBallToAdversaryGoal

Para esta primeira cena, o objetivo do *robot* era controlar a bola até à baliza adversária e marcar golo, concluindo assim com sucesso o seu objetivo. De forma a que este conseguisse controlar a bola, elaborámos uma função de fitness que tem em conta o número de vezes que o agente tocou na bola ("hitBall"), o inverso da distância que o agente se encontra da bola ("distanceToBall"), a velocidade média do *D31* ("agentSpeed") e por fim o inverso da distância à baliza adversária ("distanceToAdversaryGoal").

Ao ter em conta o número de vezes que o agente tocou na bola, podemos controlar o seu movimento atribuindo ao *robot* uma recompensa por ter conseguido tocar na bola, ficando assim mais perto de atingir o seu objetivo. De igual modo, através do uso da distância média do agente à bola e da distância média do agente à baliza adversária, podemos aumentar a aptidão do robot, dado que quanto mais perto este estiver quer da bola quer da baliza do seu adversário, maior será a probabilidade de ser bem-sucedido e, através do uso destes atributos que compõem a rede neuronal, conseguimos concluir com sucesso o objetivo da primeira cena.

Ainda para esta primeira cena, cada simulação foi gerada tendo em conta 100 gerações com um tamanho da população de 50. O valor para o tamanho do torneio ("Tournament Size") foi de 5 com uma taxa de mutação de 15% para o agente vermelho e uma taxa de recombinação de 70%.

```
1   Function GetScoreRed()
2     fitness ← 0;
3           fitness ← fitness + 0.3 * hitTheBall
4           fitness ← fitness + (1 − distanceToBall);
5           fitness ← fitness + agentSpeed;
6           fitness ← fitness + 50 * (1 − distanceFromToAdversaryGoal);
7         return fitness;
```

Fig.3 - Função de Fitness para a cena 1

4.1.1. Análise Experimental

- Gráficos com 0% probabilidade de Mutação e de 70% Crossover

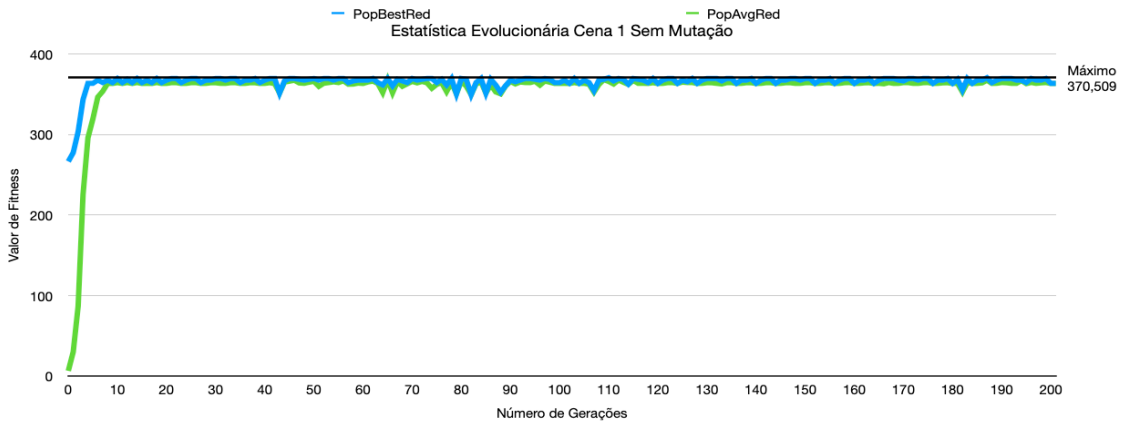


Fig.4 - Representação da Experiência Evolutiva

Através da análise da Figura 4 conseguimos observar que para a primeira simulação, o agente começa com um valor de médio de fitness bastante baixo, quase nulo, devido ao facto do processo de aprendizagem ainda se encontrar numa fase bastante introdutória. Ao gerar novas gerações, o seu valor médio também vai aumentando.

De seguida, conseguimos verificar que o agente começa a simulação com o valor máximo que se obteve na simulação anterior. Através das diferentes possibilidades e simulações este consegue chegar até a um valor de *fitness* de 370,509. Na figura 4 conseguimos observar o seu processo evolutivo ao longo da segunda simulação e, tal como seria de esperar, apresenta valores muito semelhantes entre si devido ao facto das entidades serem semelhantes uma para com as outras, não apresentando por isso um bom papel evolutivo.

- Gráficos com 15% probabilidade de Mutação e de 70% de Crossover

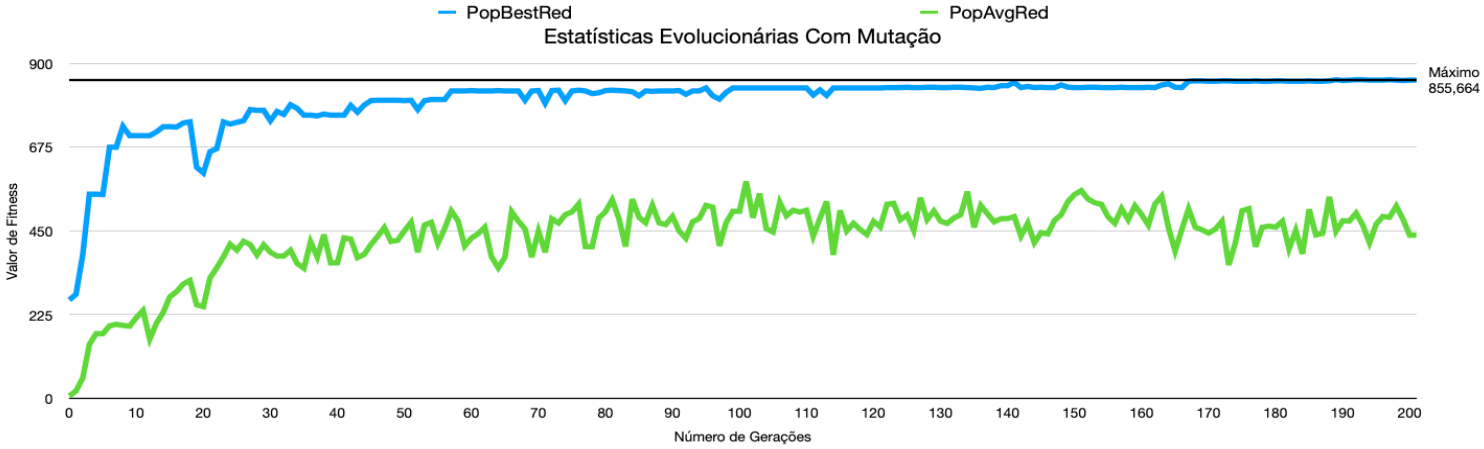


Fig.5 - Representação da Experiência Evolutiva

Dado que com valores de mutação nulos os resultados obtidos não foram os melhores devido à ausência de variação genética, decidimos implementar variabilidade genética através do operador de mutação Gaussiana.

Por garantir variabilidade genética, ou seja, garantir que indivíduos apresentem DNA diferentes, as mutações são consideradas um ponto importante para o processo de evolução, dado que é através destas que surgem novas características adaptativas (processo de aprendizagem do agente D31), o que garantem um mecanismo de seleção para aqueles indivíduos mais aptos (maior valor de aptidão/fitness).

A introdução de uma mutação não garante que o agente fique mais apto a concluir o seu objetivo pelo que a introdução de uma mutação pode proporcionar regressão do agente. Mas, como existe uma elevada variabilidade genética introduzida pela existência da mutação, existe também uma maior chance de o *robot* conseguir concluir com sucesso o seu principal objetivo de controlar a bola até à baliza adversária.

Através da análise da Figura 5, conseguimos perceber que este começa com um valor de fitness perto de 0 e à medida que vamos gerando novas combinações, este valor aumenta significativamente. Como se pode verificar, não é linear e por vezes também decresce. Tal razão é explicada em cima dado que devido à existência

de mutações, a mutação introduzida no agente pode não ser benéfica para a sua aprendizagem e daí o valor decresce, mas, como depois de realizar a experiência e verificar que o resultado obtido não foi o melhor, volta ao melhor estado que obteve até ao momento. Por fim, o valor de fitness acaba por estagnar por volta do valor máximo obtido em 833,658.

Finalmente verificamos que o agente vermelho termina a simulação com um valor de fitness de 855,664 que, comparativamente ao valor obtido na 1ª simulação corresponde a um aumento de 22,006 pontos de aptidão.

4.2 Cena 1.1: Evolving-ControlTheBallToAdversaryGoal-Random

Partindo para uma nova cena, esta é bastante semelhante à cena anterior em que a única diferença evidenciada é o posicionamento da bola. Nesta cena a bola é posicionada de forma aleatória complicando assim o trabalho do agente *D31*.

Para este mapa, para além de todos os *inputs* anteriores, temos também de considerar o número de vezes em que a bola muda de posição ao longo da simulação. Neste mapa todas as simulações foram geradas mudando a posição da bola a cada geração. Esta escolha deve-se ao facto de proporcionar ao *robot* uma aprendizagem mais lenta, ou seja, como a posição da bola não é sempre a mesma, este terá mais dificuldade em aprender dado que apesar de ter um bom valor de aptidão para uma geração não significa que a próxima geração seja igualmente boa dado que a posição da bola mudou.

Como estamos a alterar a posição da bola a cada geração, a função do cálculo da aptidão terá um papel fulcral no processo de aprendizagem do *D31*.

Inicialmente tivemos em conta a seguinte função de aptidão:

```
1  Function GetScoreRed()
2      fitness ← 0;
3      fitness ← fitness + 100 * GoalOnAdversaryGoal;
4      fitness ← fitness + 0.5 * (hitTheBall);
5      fitness ← fitness − 75 * (GoalsOnMyGoal);
6      fitness ← fitness + 50 * (1 − distanceToBall);
7      fitness ← fitness + (1 − distanceToAdversaryGoal);
8      fitness ← agentSpeed;
9  return fitness;
```

Fig.6 - Função de aptidão inicial para a cena 1.1

Através desta função, o objetivo era recompensar fortemente o agente caso este marcasse golo na baliza adversária e puni-lo caso marcasse na sua própria baliza. Como a posição da bola é aleatória, o principal objetivo do robot era ficar o mais próximo da bola possível para que depois a pudesse levar de forma controlada até à baliza adversária.

Com o uso desta função de aptidão havia dois erros importantíssimos o que não permitiam com que o agente conseguisse desempenhar o seu papel corretamente.

Caso este ficasse encurralado num canto do campo com a bola, iria receber inúmeros pontos por estar a tocar na bola e, na verdade, o seu comportamento estava errado dado que ficava preso no canto.

Na eventualidade do agente não tocar na bola e ir diretamente para a baliza adversária, este iria receber pontos dado que a sua distância à baliza adversária era muito baixa pelo que o valor de aptidão recebido era máximo para esse parâmetro. Como tal, este ficava preso na baliza adversária pensando que tinha feito um excelente trabalho dado que a sua posição era próxima da baliza do seu oponente, mas na verdade, este nem tinha tocado na bola pelo que tinha falhado o seu objetivo.

Após algumas mudanças nos valores e dezenas de simulações (em que cada simulação correspondia a 100 Gerações) melhorou-se a função de aptidão sendo esta agora composta por:

```
1  Function GetScoreRed()
2      fitness ← 0;
3      fitness ← fitness + 100 * GoalOnAdversaryGoal;
4      fitness ← fitness + (hitTheBall);
5      fitness ← fitness − 75 * (GoalsOnMyGoal);
6      fitness ← fitness + 50 * (1 − distanceToBall);
7      fitness ← fitness + agentSpeed;
8      fitness ← fitness + 50 * (1 − distanceFromBallToAdversaryGoal);
9  return fitness;
```

Fig.7 - Função de aptidão final para a cena 1.1

A grande diferença para a função original é que agora o agente é mais recompensado por tocar na bola e recebe agora mais pontos caso a distância da bola à baliza adversária seja mínima sinal que está mais perto de marcar golo na baliza adversária. Nesta função foi removida a pontuação do agente caso este se encontrasse muito perto da baliza adversária ("distanceToAdversaryGoal") de forma a combater o problema encontrado e descrito em cima.

4.2.1. Análise Experimental

- Gráfico com a 1ª função de fitness

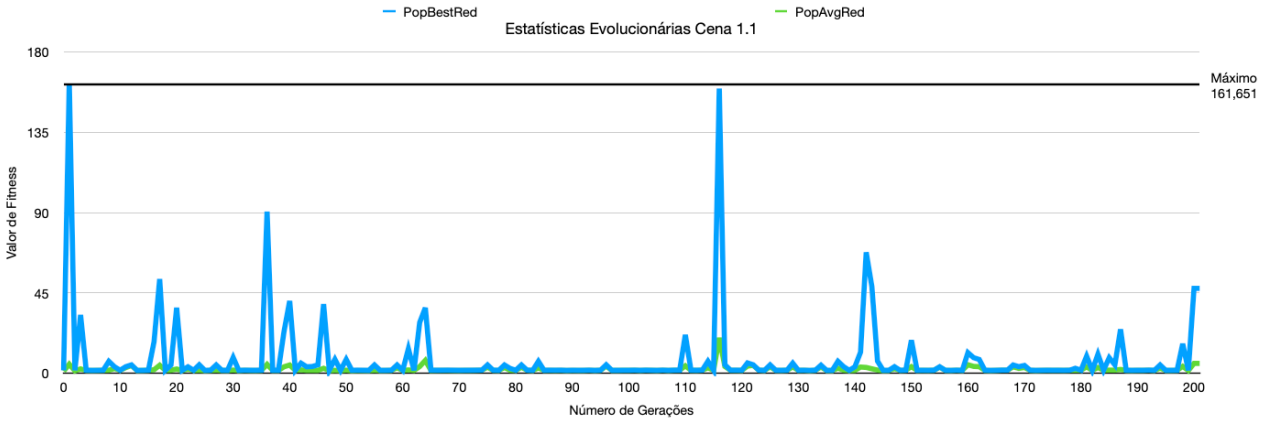


Fig.8 - Representação da Experiência Evolutiva

Através da análise dos resultados da Figura 8, conseguimos visualizar que no início este tem um excelente desempenho dado que atinge o seu valor máximo, mas nas gerações seguintes, este valor decresce drasticamente devido à natureza da cena. como se trata de uma posição aleatória da bola, o desempenho e a capacidade de aprendizagem do agente serão menores e daí a necessidade de adaptar a função de fitness para uma que valorizasse a distância do agente à bola e a distância da bola à baliza adversária de forma tentar que o processo de aprendizagem se tornasse mais eficaz.

A criação de outra função de aptidão também se deve ao facto de ao analisar o valor médio de fitness ao longo da primeira e segunda simulação, este não tem alterações significativas o que se traduz numa aprendizagem lenta sem grande alteração.

- Gráfico com a 2ª função de fitness

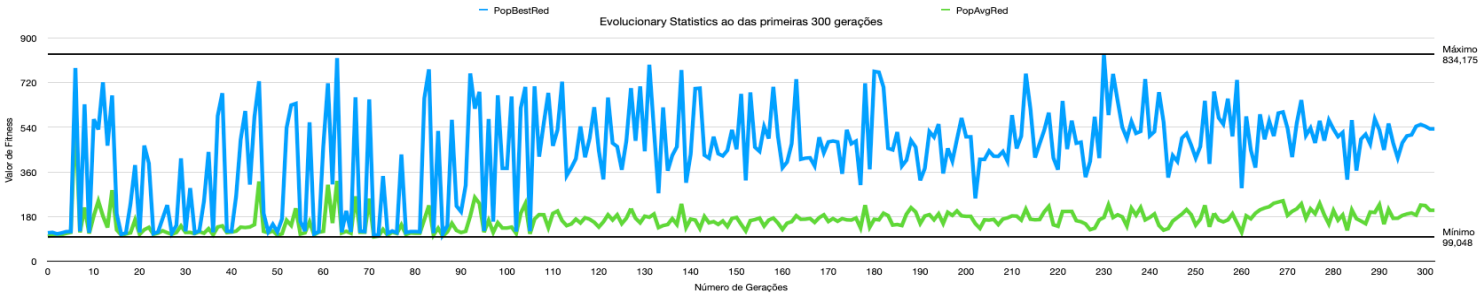


Fig.9 - Representação da Experiência Evolutiva

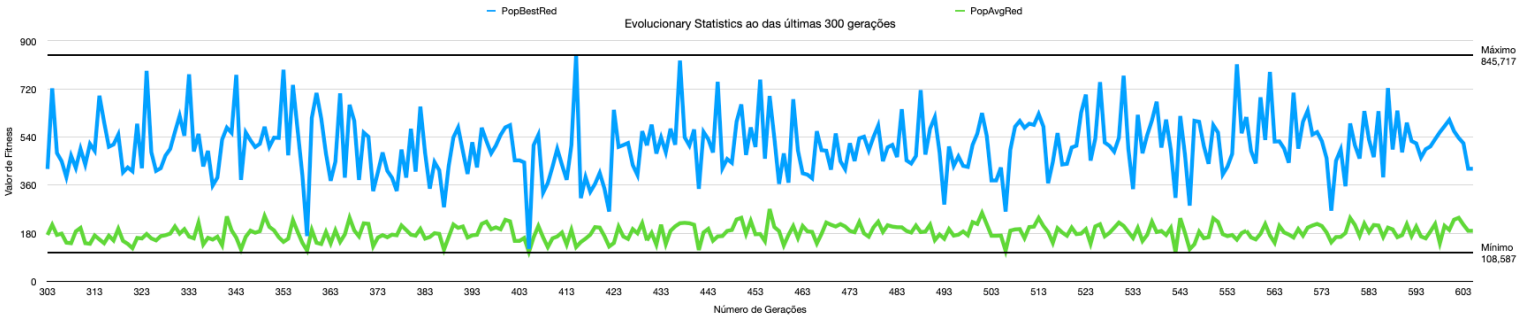


Fig.10 - Representação da Experiência Evolutiva

Fazendo uso da nova função de aptidão, através do gráfico 10, conseguimos verificar que o agente se comporta de forma estranha dado que ora está com bom valor de fitness ora está com um valor perto de 0. Tais resultados podem ser explicados devido ao facto de ser a primeira simulação do agente, logo este não sabe como se comportar perante a posição da bola. À medida que vamos evoluindo no número de gerações, o valor de fitness obtido começa a ficar mais regular devido ao processo de aprendizagem do agente. Este começa a aprender como se deve comportar e, já não comete tantos erros como cometia no início da simulação.

Após a geração 300, o robot continua a aprender e acaba por melhorar o seu valor de fitness. Este valor apesar de constituir uma melhoria para o comportamento do agente, não é o suficiente para concluir com sucesso este mapa. Deste modo, após 600 gerações, o D31 apesar de ir ao encontro com a bola, não a consegue levar de forma controlada para a baliza do seu adversário.

4.3 Cena 2: Evolving-Defense

Na segunda cena, o objetivo é defender um remate da bola. Para este caso, não foram necessárias muitas evoluções, visto que o agente se tinha de posicionar de tal maneira a que protegesse a sua baliza de um remate fixo. Por causa disso, foi possível chegar a uma solução rapidamente. Isto deve-se ao facto da previsibilidade do remate que fez com que o agente percebesse rapidamente a sua origem. Para tal, teve de se ter em conta quatro informações distintas: a quantidade de vezes que toca na bola (hitTheBall), a distância média à bola (distanceToBall), a distância da bola à baliza do agente (distancefromBallToMyGoal) e a quantidade de golos sofridos (GoalsOnMyGoal). A estratégia pensada gira à volta do agente bater na bola, evitar que ela entre na sua baliza e imobilizá-la num canto

A função de fitness tem então a seguinte forma:

```
1   Function GetScoreRed()
2     fitness ← 0;
3     fitness ← fitness + 20 * hitTheBall;
4     fitness ← fitness − 100 * (average(DistanceToBall));
5     fitness ← fitness + 10 * (average(DistancefromBallToMyGoal));
6     fitness ← fitness − 100 * GoalsOnMyGoal;
7     if GoalsOnMyGoal = 0 then
8       fitness ← fitness + 200;
9     end
10  return fitness;
```

Fig.11 - Função de Fitness inicial do cenário *Evolving-Defence*

Como se pode reparar, tivemos como abordagem garantir que o agente tocasse na bola (para realizar o ato de defender) e obrigá-lo a reduzir a sua distância à bola e desta à sua baliza. Finalmente, era fortemente penalizado por cada golo que sofresse e vigorosamente recompensado caso não sofresse nenhum golo.

Finalmente, para gerar os agentes reunimos as seguintes configurações: 300 gerações com um tamanho de população de 50. Devido aos resultados estudados anteriormente, optámos pelo valor 4 para o tamanho do torneio ("Tournament Size") com uma taxa de mutação de 12% para o agente vermelho e uma taxa de recombinação de 70%. Note-se que o período de evolução de cada simulação foi colocado a 8 segundos já que o objetivo era apenas defender.

4.3.1. Análise Experimental

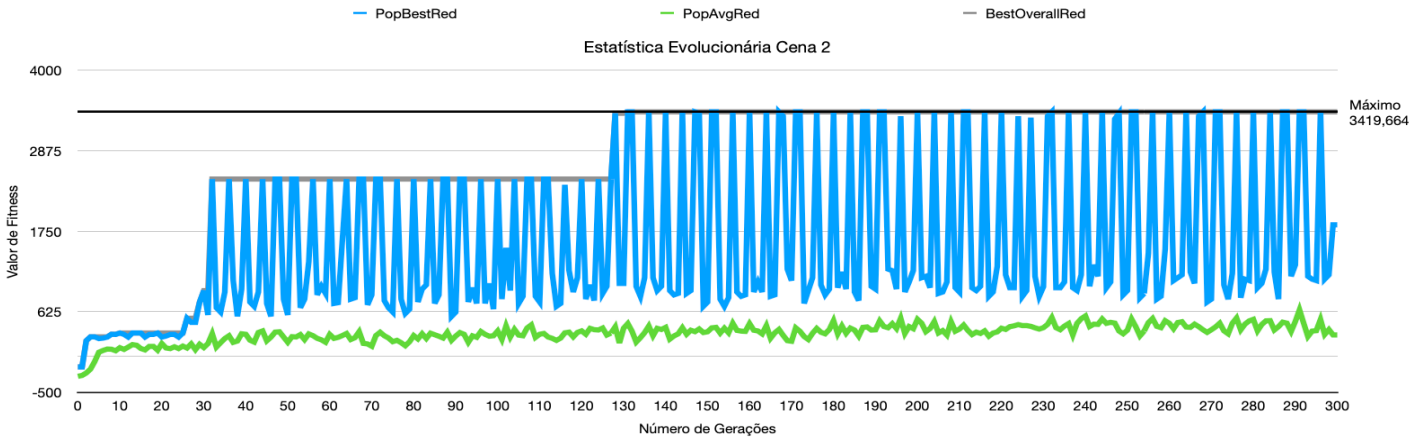


Fig.12 - Representação da Experiência Evolutiva

Como se pode observar, a população apresenta um desempenho negativo nas primeiras 6 gerações. Após alguma evolução, pela geração 36, a média dos agentes torna-se acima dos 200 (que é a pontuação recebida caso o agente não sofra nenhum golo) e todos conseguiam defender consistentemente a bola. Ao longo da execução, registaram-se alguns comportamentos diversos como por exemplo:

• Contrariar a direção da bola e marcar golo na baliza adversária. Assim, era gerado um novo remate que prejudicava a sua performance. Para resolver esta situação, o agente passou apenas a contrariar levemente o remate inicial e saía de perto da bola.

• Por causa do peso da distância à bola e da distância da bola à baliza do agente, este prendia a bola num dos cantos do seu lado do campo e ficava a rematá-la num sítio preso.

Vale a pena referir a discrepância dos pontos em ambos os gráficos. Isto deve-se ao facto da quantidade de pontos atribuídos na função de fitness para cada toque de bola. Se o agente acabasse por prender a bola num canto então ia tocar-lhe até acabar o tempo de simulação, acumulando muitos pontos. Para obrigar todos os agentes a realizar a mesma estratégia podia-se aumentar a pressão seletiva, ou seja, aumentar o tamanho do torneio, fazendo com que os agentes referidos anteriormente sejam utilizados mais frequentemente no crossover.

Para a versão aleatória, o agente teve muitas dificuldades em conseguir defender consistentemente a bola, mesmo gerando 1200 gerações, foram obtidos resultados muito inconsistentes. Como o agente não conseguia prever a direção do remate tentava defender de uma localização qualquer daí apresentar resultados aceitáveis de vez em quando. Como tal, o gráfico da experiência das últimas 300 gerações (900-1200) é o que se segue:

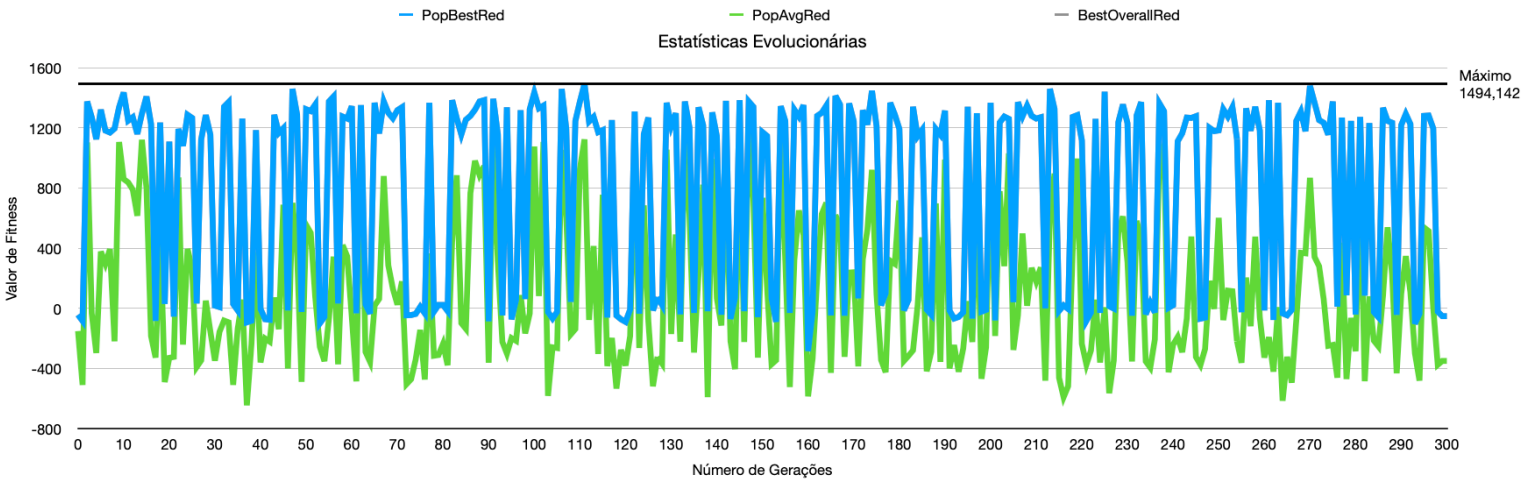


Fig.13 - Representação da Experiência Evolutiva

Note-se que tanto o "PopBestRed" como o "BestOverallRed" acabam por tomar o mesmo valor. Uma análise breve do gráfico indica que o agente só se encontra preparado para algumas situações e que a média acaba por se tornar negativa muitas vezes. Para contornar esta situação decidiu-se repensar a função de fitness e incorporar outra estratégia que desse prioridade à distância do agente da bola e não aos toques de bola.

Assim, a função de fitness foi melhorada para:

```
1   Function GetScoreRed()
2     fitness ← 0;
3     fitness ← fitness + 3 * hitTheBall;
4     fitness ← fitness + 85 * (1 / (10 * average(DistanceToBall)));
5     fitness ← fitness + 40 * (average(DistancefromBallToMyGoal));
6     fitness ← fitness − 200 * GoalsOnMyGoal;
7     if GoalsOnMyGoal = 0 then
8       fitness ← fitness + 1000;
9     end
10  return fitness;
```

Fig.14 - Função de Fitness nova do cenário *Evolving-DefenceRandom*

Mesmo assim, não se obteve resultados relevantes pelo que optámos em aumentar a pressão seletiva, passando o tamanho do torneio "Tournament Size" para 7. Obtendo os seguintes resultados (geração 300-600):

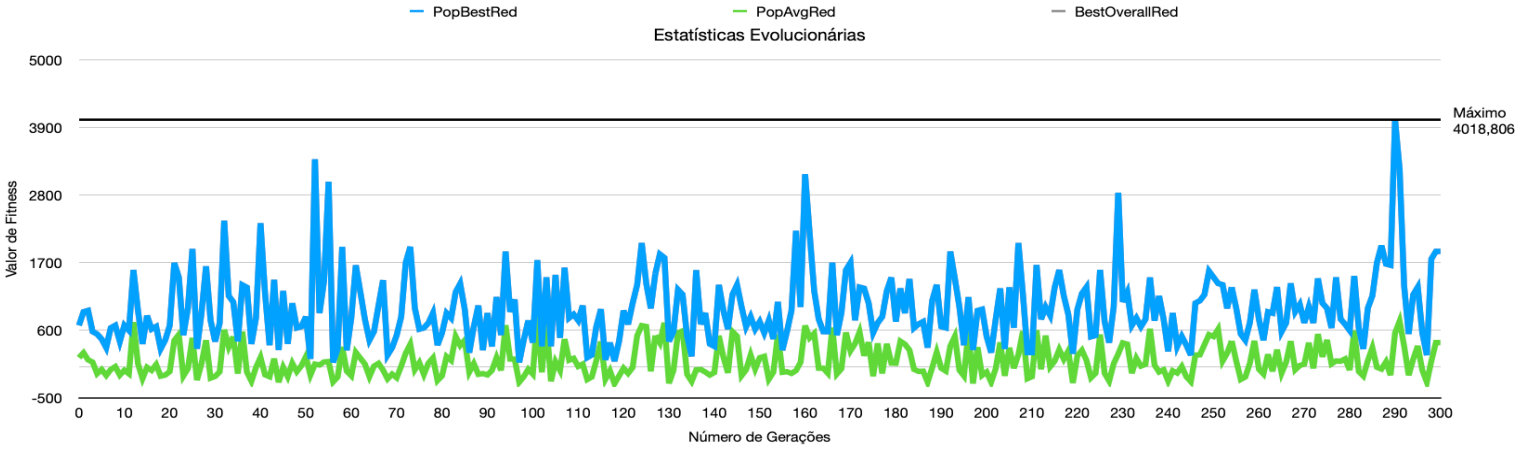


Fig.15 - Representação da Experiência Evolutiva

É possível notar algumas melhorias significativas, a média do agente agora é mais próxima da média (devido à maior pressão seletiva) todavia o comportamento do agente não era consistente devido ao fator aleatório imposto por esta cena. O comportamento seguia a seguinte regra: caso não conseguisse defender à primeira, este posicionava-se de modo a que conseguisse defender os próximos remates. Após alguma experimentação, percebemos que existiam algumas soluções possíveis que podíamos implementar. Posicionar o agente de modo a que ficasse constantemente no centro da baliza, defendendo todos os remates, independentemente da direção, ou então fechar imediatamente a distância do agente à bola, garantindo que, ao começar a simulação, este se dirigia sempre contra a bola, de modo a fechar a linha de remate e defender a bola.

Então, foi criada uma nova função de fitness:

```
1   Function GetScoreRed()
2     fitness ← 0;
3     fitness ← fitness + 85 · (1 / (10 · average(DistanceToBall)));
4     fitness ← fitness − 200 · GoalsOnMyGoal;
5     if GoalsOnMyGoal = 0 then
6       fitness ← fitness + 1000;
7     end
8   return fitness;
```

Fig.16 - Função de Fitness melhorada do cenário *Evolving-DefenceRandom*

Deu origem aos seguintes resultados (geração 600-900):

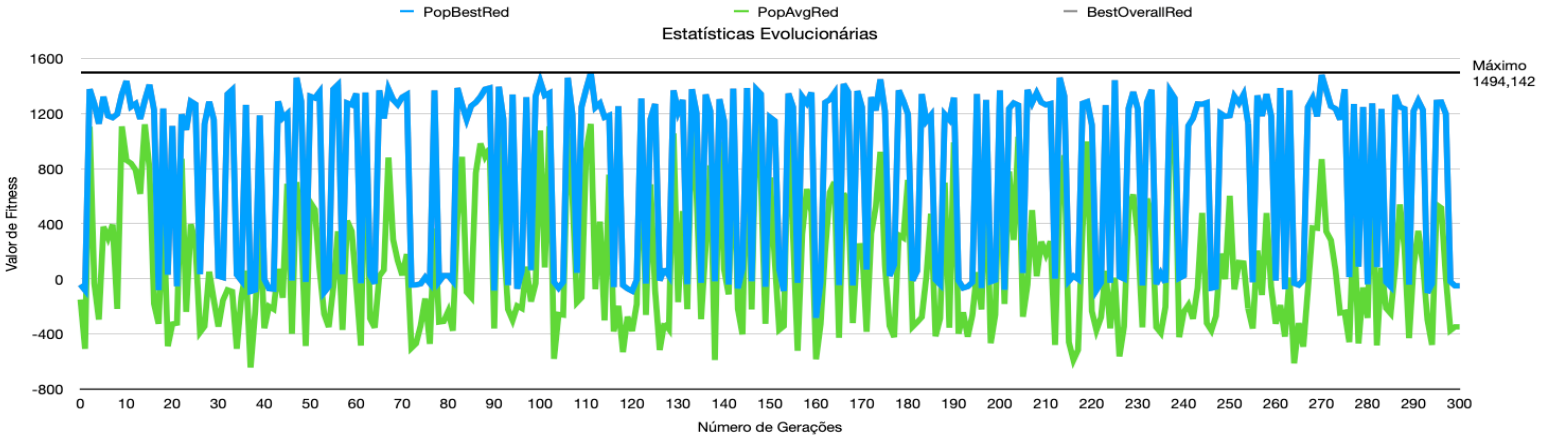


Fig.17 - Representação da Experiência Evolutiva

Finalmente, o agente conseguia defender a bola de maneira mais consistente, mesmo apresentando algumas dificuldades para casos onde não conseguia prever o remate da bola. Mesmo assim, para cada geração havia quase sempre pelo menos um agente que conseguia defender a bola com sucesso, sem nunca sofrer nenhum golo. Tal razão para esta inconsistência deve-se ao fator aleatório imposto nesta cena. O agente pode aprender a defender um remate vindo de baixo, mas na geração seguinte não pode usar essa mesma estratégia para defender um remate vindo de cima.

Para uma outra experiência seguimos a seguinte filosofia: aplicar esta nova função de fitness à condição inicial (remates não aleatórios) e avaliar a evolução do agente (note-se que o tamanho do torneio foi aumentado para 5, antes era 4):

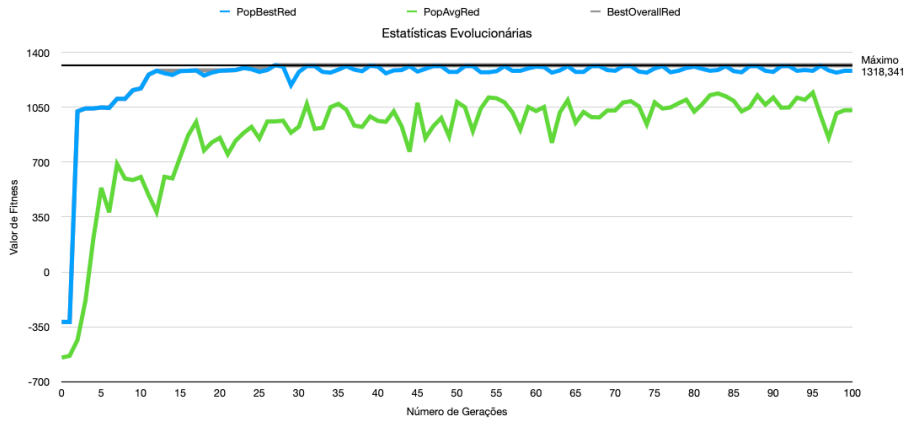


Fig.18 - Representação da Experiência Evolutiva

Foram necessárias apenas 100 gerações para obter resultados extremamente satisfatórios, uma média acima de 1000 implica que a maior parte da população passou sem ter sofrido golos. Conclui-se então que aumentar a pressão seletiva teve um impacto positivo no comportamento do agente, bem como ajustar a função de fitness de modo a que priorizasse diminuir a sua distância à bola. Relativamente ao primeiro gráfico apresentado (Fig. 12), nota-se grande melhoria relativamente ao *fitness* que agora é mais constante e uniforme. Isto deve-se ao facto de não usar o estímulo *hitTheBall* dado que, nos casos onde o agente empurra a bola contra o canto, o seu score ia aumentar abruptamente porque ia tocar muitas vezes na bola. Para além disso, a média agora está muito mais próxima do *best* (e está acima de 1000 pontos), o que implica que em média, todos os agentes defendem a bola com relativa facilidade. Em termos de estratégias, este desenvolveu as mesmas referidas anteriormente: Ir contra a bola, defendendo e afastando-se da mesma, ou então dominá-la para um canto.

Para uma última experiência, optámos por correr as configurações anteriores, mas sem mutação, para avaliar a evolução do comportamento do agente. Os resultados são os que se seguem:

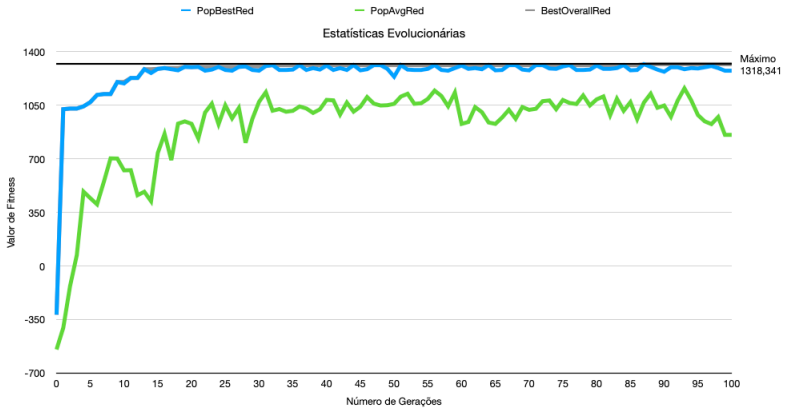


Fig.19 - Representação da Experiência Evolutiva

Como se pode analisar, mesmo não introduzindo informação genética nova, os agentes conseguiram evoluir de modo a que a média se mantivesse à volta dos 1000 pontos. Mesmo assim, não manter diversidade genética inibe a evolução do indivíduo, tal como foi discutido anteriormente.

4.4 Cena 3: Evolving-One vs One

Para o cenário final, o objetivo passa de evoluir apenas um agente para dois agentes em simultâneo, em que o seu meio de aprendizagem é em competição um com o outro. Para este mapa foram desenvolvidas várias funções de aptidão baseadas em algumas anteriores, e durante as execuções foi possível observar que sempre que um dos agentes era sempre mais apto que o outro, quer fosse mais rápido ou estivesse mais perto da bola, acabava sempre ou por quase marcar golo ou por marcar mesmo. Isto muda, porém, quando colocamos o best azul contra o vermelho no final de várias gerações, como ambos evoluíram segundo a mesma função de fitness, os comportamentos são bastante semelhantes e acabam muitas vezes por não conseguir marcar golo.

Cada função teve números de gerações diferentes, porém as configurações eram semelhantes: 30 segundos de simulação para poder-se desenvolver bem o comportamento de uma situação 1vs1, 50 de tamanho de população, 5 de tamanho de torneio, 15% de probabilidade de mutação e 70% de recombinação, para ambos o azul e o vermelho.

```
1   Function GetScoreRed()
2       fitness ← 0;
3       fitness ← fitness + (1 − distanceToBall)
4       fitness ← fitness + (1 − distanceFromBallToAdversayGoal)
5       fitness ← fitness − (1 − distanceFromBallToMyGoal)
6       fitness ← fitness + agentSpeed
7       fitness ← fitness + abs(fitness * 0.5)*GoalsOnAdversary
8       fitness ← fitness + abs(fitness * 0.5)*GoalsOnMyGoal
9       fitness ← fitness − hitTheWall
10  return fitness
```

Fig.20 - 1ª Função de aptidão para a cena 3

Esta primeira função de aptidão recompensa agentes que mantenham pouca distância à bola e da bola à baliza, que atinjam velocidades mais elevadas, e como é óbvio, que marquem golos. Pelo contrário, penaliza agentes que deixem a bola aproximar-se da sua baliza, para que evitem sofrer golos, penaliza quando sofrem golos e quando tocam nas paredes.

Com esta função o comportamento mais evoluído de um agente foi dirigir-se rapidamente em direção à bola, "rematando" assim para a baliza adversária, isto era esperado visto que não se inclui na função os toques na bola.

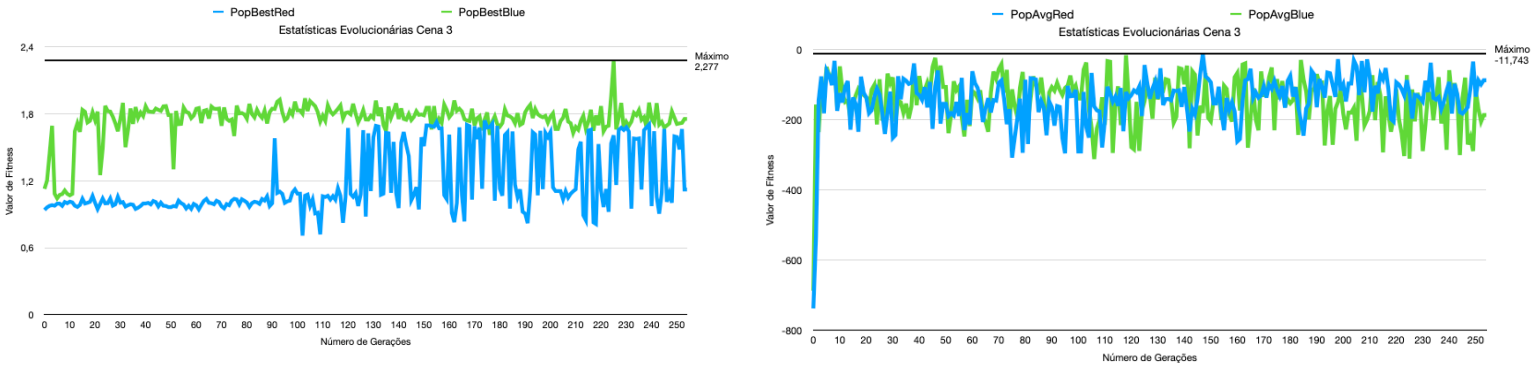


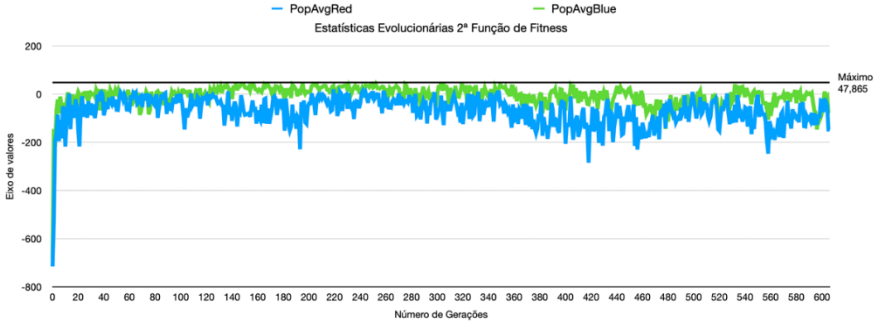
Fig.21 - Estatística Evolutiva para a cena 3

Ao fim de 250 gerações, analisando os gráficos é possível concluir que esta função não possui grande potencial evolutivo tendo pouca ou nenhuma evolução de geração em geração. Também é possível observar uma evolução desigual entre os dois agentes pelo gráfico, e também com a execução com o best de cada cor observava-se que o vermelho agia mais passivamente e apenas tentava não sofrer golo e manter distâncias baixas à bola, daí os valores mais baixos de aptidão.

```
1   Function GetScoreRed()
2       fitness ← 0;
3       fitness ← fitness + (1 − distanceToBall)
4       fitness ← fitness + (1 − distanceFromBallToAdversayGoal)
5       fitness ← fitness − (1 − distanceFromBallToMyGoal)
6       fitness ← fitness + agentSpeed
7       fitness ← fitness + abs(fitness * 2) * GoalsOnAdversaryGoal
8       fitness ← fitness + abs(fitness * 2) * GoalsOnMyGoal
9       fitness ← fitness + (hitTheBall * 0.1)
10      fitness ← fitness − hitTheWall
11  return fitness
```

Fig.22 - 2ª Função de aptidão para a cena 3

A segunda função usada foi baseada na primeira, mas com algumas alterações: aplicou-se uma fórmula que aplicava mais peso nos golos marcados e sofridos, e passou-se a somar 10% dos toques na bola para incentivar o agente a controlar melhor a mesma.



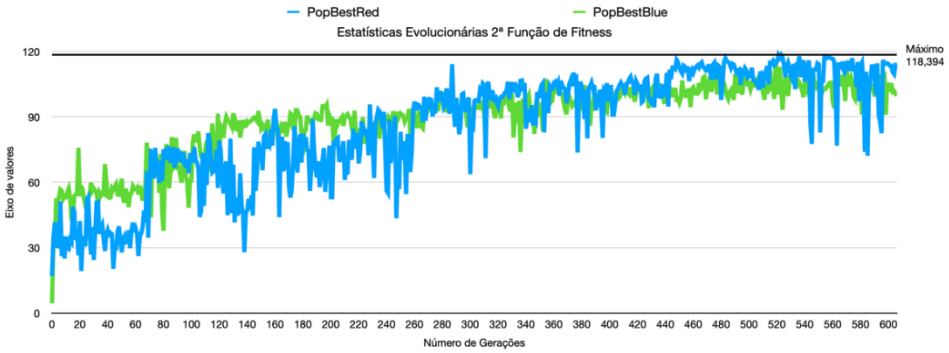


Fig.23 - Estatísticas Evolucionárias Cena 3

Analisando o primeiro gráfico pode-se concluir facilmente que esta função fornece aos agentes melhor capacidade de aprendizagem que a anterior, visto que demonstra um crescimento contínuo durante cerca de 500 gerações, só começando a abrandar depois. Também se observa que as evoluções dos agentes são semelhantes ao contrário da função anterior.

O comportamento alcançado ao fim de 600 gerações foi semelhante ao cenário de controlar a bola até à baliza adversária, porém com oposição por parte de outro agente, oposição esta que se fosse um indivíduo igualmente apto, ambos acabavam a empurrar a bola um contra o outro para tentar chegar à baliza, não sucedendo nesta tentativa.

Finalmente, com esta mesma função foram geradas 700 gerações, mas com a posição da bola a variar a cada 5 gerações. Os resultados obtidos não foram muito impressionantes, sendo os agentes incapazes de se ajustar à mudança de posição da mesma.

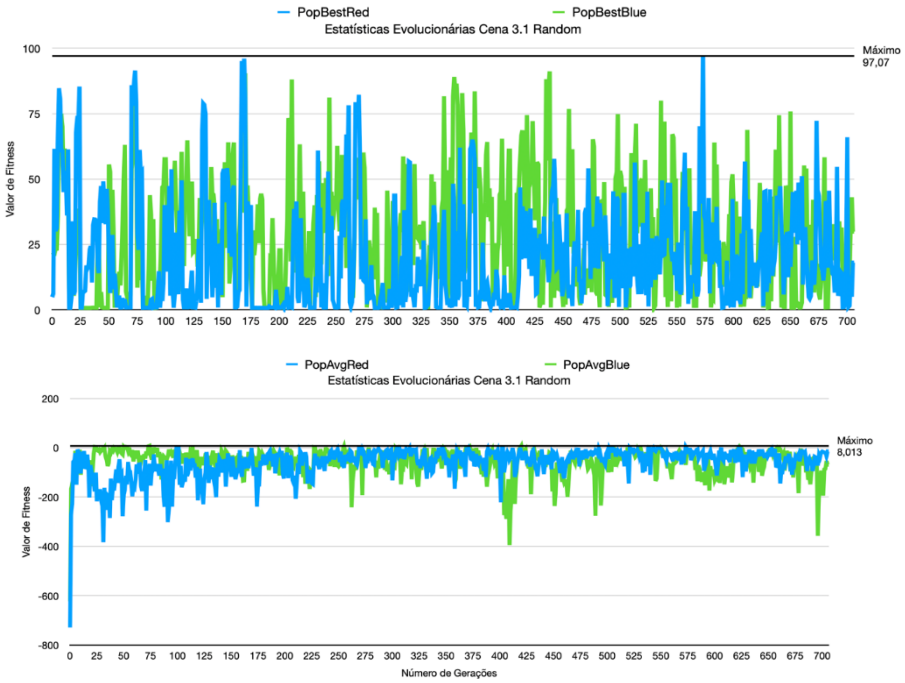


Fig.23 - Estatísticas Evolucionárias Cena 3.1 Random

É possível observar que com a posição da bola a alterar a cada 5 gerações, a evolução dos agentes foi bastante dificultada (com esta função), os valores oscilam bastante entre 0 e perto de 100, correspondendo os picos a situações em que a bola se encontrava mais perto de um agente e/ou central em relação ao campo.

Para obter melhores resultados com essa configuração seria necessário ou adaptar a função ou gerar significativamente mais gerações, visto que a evolução com a bola a mudar de posição é muito mais lenta e desafiadora para os agentes.

5 Discussão

Entre as várias experiências realizadas, conseguimos realizar com sucesso os cenários propostos, sendo que as versões aleatórias demonstraram ser um desafio maior. Isto deve-se à imprevisibilidade da bola, ou seja, se o agente aprender a rematar/defender uma bola numa dada posição, na geração seguinte terá dificuldades em fazer o mesmo, caso as condições iniciais se alterem. Outra conclusão importante deve-se à importância da pressão seletiva e à mutação. Estes atributos permitem, respetivamente, uniformizar o comportamento dos agentes e selecionar o melhor de maneira mais consistente para a recombinação e introduzir variedade genética na população.

6 Conclusão

A realização deste estudo permitiu a criação de controladores que ajudaram o agente D31 a desempenhar várias tarefas. Aquando da sua execução, foram reunidos vários dados que permitiram sintetizar a importância da rede neuronal, aliada a um algoritmo genético. Finalmente, deu a entender que a possibilidade de evolução é imensa e que, de facto, se podem gerar indivíduos complexos e capazes de superar vários cenários, tudo graças à inteligência artificial.